UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

IMPLEMENTACIÓN DE ALGORITMOS ESTÉREO DENTRO DE LA PLATAFORMA ROS

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Abdoulaye Diallo

Tutor: Pablo Marín plaza

Leganés, Septiembre 2014

Índice

Índice de Figuras

Índice de Tablas

Listado de Acrónimos

TFG Trabajo Fin de Grado.

UC3M Universidad Carlos III de Madrid.

IVVI Intelligent Vehicle based on Visual Information (Vehículo Inteligente basado en Información Visual).

ROS Robot Operating System (Sistema Operativo Robótico).

API Application Programming Interface (Interfaz de programación de aplicaciones).

SDK Software Development Kit (kit de desarrollo de software).

OpenCV Open Source Computer Vision (Biblioteca libre de visión artificial).

UDP User Datagram Protocol (protocolo del nivel de transporte basado en el intercambio de datagramas).

SAD Sum of Absolute Differences (suma de diferencias absolutas).

Point Grey Empresa fabricante de cámaras digitales.

Triclops Software de Point Grey para procesamiento de imágenes.

Agradecimientos

En primer lugar quiero expresar mi gratitud y mi reconocimiento hacia toda mi familia, que a pesar de la lejanía, siempre me han apoyado y han confiado en mí.

También debo dar las gracias a todos los amigos y compañeros de la Universidad Carlos III, por trabajar juntos y por compartir estos años conmigo.

Y por supuesto, agradezco enormemente a mi tutor, quien me ha guiado en la realización de este trabajo, por sus ayudas y sus consejos, así como a todos los profesores que han contribuido en mi formación académica.

Resumen

En este proyecto se trabaja en el área de la visión artificial, una disciplina que extrae descripciones tridimensionales de los elementos de una escena con la ayuda de procesos de tratamiento realizados en las imágenes adquiridas con cámaras.

Los objetivos de la visión artificial son varios y uno de ellos es la determinación de la distancia a los objetos. Esto se puede conseguir utilizando la visión estéreo con un par de cámaras previamente calibradas.

El objetivo de este proyecto es la obtención de imágenes correctamente rectificadas de una de las cámaras de visión estéreo que se pretende incorporar en la plataforma IVVI 2.0 del laboratorio LSI (Laboratorio de sistemas inteligentes) de la universidad Carlos III (un vehículo de investigación para la implementación de sistemas basados en visión por computador, con el objetivo de implementar Sistemas Avanzados de Ayuda a la Conducción **[1]**) y publicar estas imágenes en ROS, concretamente se trata de la cámara Bumblebee xb3 de Point Grey.

Principalmente se trabaja en las dos primeras fases de la visión estéreo, el pretratamiento y el matching para comprobar la correcta rectificación de las imágenes mediante el análisis de los mapas de disparidad a partir de dos imágenes estéreo. Para ello se realiza la rectificación de las imágenes y se sacan los mapas de disparidad con OpenCV (class stereoBM).

También se realiza la rectificación de las imágenes y se sacan los mapas de disparidad con el software del fabricante (triclops), se realiza una comparación de los dos métodos de rectificación y se decide realizar el procesamiento de las imágenes con el software del fabricante.

Por último, debido al método de procesamiento elegido y la incompatibilidad de las librerías Triclops / ROS, se ha tenido que utilizar la técnica de memoria compartida para poder publicar las imágenes rectificadas en ROS.

Abstract

This project focuses on the area of computer vision, a discipline that uses image processing to extract features of three-dimensional elements in a given scene.

There are several objectives of computer vision. One of those objectives is determining the distance to objects that can be achieved using stereo vision with a pair of previously calibrated cameras.

The goal of this project is to obtain properly rectified images from one of the stereo vision cameras, Bumblebee xb3 camera of Point Grey, so that it can be endorsed in the IVVI car of the Intelligent Systems Laboratory, Universidad Carlos III (a research project with the aim of implementing a vehicle´s Advanced Driving Assistance Systems based on vision systems **[1]**) and publish these images in ROS.

The project is based mainly on the first two phases of the stereo vision, pre-processing and matching to verify proper image rectification by analyzing the disparity maps from two stereo images. To carry out these phases, image rectification and maps disparities are performed with OpenCV (class stereoBM).

Image rectification and map disparity are also performed with the Point Grey´s software (Triclops), a comparison of the two rectification methods is done and the Triclops method was selected for image processing.

Finally, given the Triclops processing method chosen earlier, it was necessary to program a shared memory to publish rectified images in ROS, as ROS and Triclops libraries are not compatible.

1 INTRODUCCIÓN

1.1 Motivación y objetivos

La visión artificial o visión por computador, es un subcampo de la **"Inteligencia Artificial"** que permite la obtención, procesamiento y análisis de cualquier tipo de información especial obtenida a través de imágenes digitales, mediante la utilización de las técnicas adecuadas.

Es la disciplina que busca replicar el sistema perceptivo humano en una máquina, extrayendo unas descripciones tridimensionales de los elementos de la escena con la ayuda de procesos de tratamiento de imagen.

Las líneas de investigación en esta área son muchas debido a las grandes demandas de un amplio espectro de aplicaciones, tales como la medicina, la industria aeroespacial, procesos industriales, los videojuegos, etc.

Los objetivos típicos de la visión artificial incluyen:

- La determinación de la distancia a los objetos.
- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes (por ejemplo, caras humanas).
- Registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.
- Seguimiento de un objeto en una secuencia de imágenes.
- Mapeo de una escena para generar un modelo tridimensional de la misma; este modelo podría ser usado por un robot para navegar por el entorno.
- Estimación de las posturas tridimensionales de humanos.
- Búsqueda de imágenes digitales por su contenido.

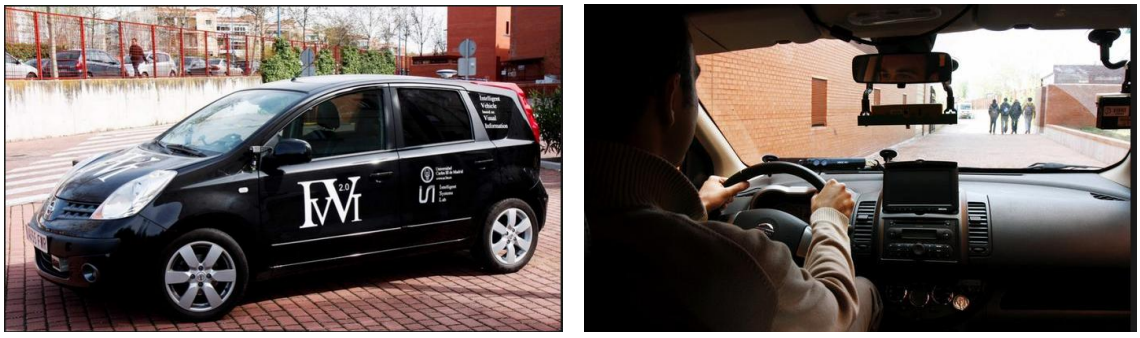
Concretamente, en la industria del automóvil la visión artificial tiene gran interés y ha aumentado mucho la investigación en esta área en los últimos años debido al gran número de accidentes de tráfico que se están produciendo.

Para mejorar la seguridad en la conducción, lograr una conducción automática y sobre todo para salvar vidas, se están desarrollando Sistemas Avanzados de Ayuda a la Conducción (Advanced Driver Assistance Systems **[2]**).

Tener la posibilidad de dotar a los vehículos de visión artificial, es muy positivo ya que permite la automatización de diversos procesos, posibilitando así la conducción autónoma y la reducción de accidentes de tráfico.

La plataforma IVVI 2.0 es un vehículo de investigación para la implementación de sistemas basados en visión por computador, con el objetivo de implementar Sistemas Avanzados de Ayuda a la Conducción.

En las imágenes de la figura 1 se puede observar el coche IVVI 2.0 del Laboratorio LSI de la Universidad Carlos III así como algunos sistemas de visión artificial que lleva incorporado:



a) Coche IVVI por fuera *b) Coche IVVI con visión artificial*

Figura 1: Coche IVVI de la UC3M ***[3]***

Objetivos:

El objetivo de este proyecto es la obtención de imágenes correctamente rectificadas de una de las cámaras de visión estéreo que irá incorporado en el coche IVVI mencionado anteriormente y publicar estas imágenes en ROS, concretamente se trata de la cámara Bumblebee xb3 de Point Grey.

Las principales etapas de la visión estéreo son:

- Pre tratamiento
  - Calibración
  - Adquisición
  - Rectificación
- Matching (emparejamiento)
- Reconstrucción (triangulación)

Este trabajo se centra principalmente en el pretratamiento y en el matching para comprobar la correcta rectificación de las imágenes mediante el análisis de los mapas de disparidad a

partir de dos imágenes estéreo. Para ello se tratará de rectificar las imágenes y sacar los mapas de disparidad mediante dos métodos, el primero con OpenCV (class stereoBM) y publicar al mismo tiempo toda esta información en ROS.

El otro método es la rectificación con triclops, que es el software del fabricante, se hará una comparación de los dos métodos de rectificación. Se elegirá el método de los triclops y se realizará una memoria compartida para poder publicar las imágenes rectificadas en ROS.

1.2 Estructura del documento

Esta memoria se explica de manera estructurada, mediante la división en capítulos, aspectos importantes que tienen que ver con el desarrollo del trabajo. En el capítulo 1 se presenta una introducción a la visión artificial así como los objetivos a conseguir en el trabajo.

El capítulo 2 está dedicado al estado de arte de la visión artificial, realizando un estudio general de las técnicas de reconstrucción 3D, los distintos métodos y en particular el método de la visión estereoscópica, después se realiza una descripción de la cámara utilizada así como sus especificaciones técnicas y también una descripción de los programas utilizados. El capítulo 3 se titula generación de mapas de disparidad y se divide en: Mapas de disparidad con OpenCV en ROS, donde se presentan los resultados obtenidos y una conclusión al final; y Mapas de disparidad con driver privativo Triclops, donde al igual que antes, se presentan los resultados y una conclusión al final.

En el capítulo 4 titulado Bumblebee xb3 con Triclops en ROS, se analiza la compatibilidad de las librerías de estos programas. En el capítulo 5 se presenta la necesidad de programación de una memoria compartida así como la publicación de las imágenes en ROS. En el capítulo 6 se presentan las conclusiones y posibles trabajos futuros y por último se presenta el presupuesto del proyecto en el capítulo 7. Al final del documento se facilita la bibliografía utilizada en este trabajo.

2 ESTADO DEL ARTE

La visión artificial es la disciplina que busca replicar el sistema perceptivo humano en una máquina. Uno de sus enfoques de investigación se basa en extraer descripciones tridimensionales de los elementos de la escena con la ayuda de procesos de tratamiento realizados en las imágenes adquiridas con una o varias cámaras.

En este capítulo, se da una vuelta por la literatura científica del tema principal del trabajo. Dos grandes bloques estructuran este capítulo: el primero es el desarrollo de un estado de arte de la reconstrucción 3D en general desde un punto de vista de la visión artificial presentando principalmente la visión estéreo. El segundo se centra principalmente en enfoques de modelado en 3D mediante la visión estereoscópica.

2.1 Técnicas de reconstrucción 3D

La reconstrucción 3D es el conjunto de técnicas que se utilizan para representar una escena real en tres dimensiones utilizando la visión por computador. Esta tecnología presenta particular interés en los últimos años debido a las numerosas solicitudes en aplicaciones.

Varias técnicas ópticas se pueden utilizar para reconstruir una escena real en 3D. Estas técnicas se clasifican principalmente en dos categorías:

Métodos activos y métodos pasivos:

Métodos de reconstrucción 3D (Métodos ópticos)	
Métodos activos	**Métodos pasivos**
- Time of flight (tiempo de vuelo) - Triangulación láser - Luz estructurada - Luz modulada - Diferencia de fase	- **Estereoscópicos** - **Silueta** - **Con ayuda del usuario (movimientos)**

Tabla 1: Técnicas de reconstrucción 3D.

2.1.1 Visión activa

El método de la visión activa es un método donde la cámara se combina con una fuente de luz con el fin de obtener las coordenadas 3D de los puntos emitidos por esta fuente en las superficies visibles de los objetos en la escena. En otras palabras, estos métodos adquieren la profundidad de una escena a partir de una fuente de luz controlada (estructurada). Diferentes tipos de fuentes de luz pueden ser utilizados, siendo los más comunes el láser y el infrarrojo.

A menudo, estos métodos presentan un aumento de complejidad de sus sistemas y presentan algunos inconvenientes tales como la exploración lenta o la degradación mecánica lo que hace que su uso en algunos contextos sea menos práctico. Los métodos activos sufren también de una cierta sensibilidad a diferentes tipos de iluminación ambiental y la reflectividad de los objetos de la escena. Para un estudio más profundizado, una buena revisión de estos métodos está propuesta por *Besl* **[4]**.

2.1.2 Visión pasiva

La visión pasiva utiliza solamente datos de imágenes adquiridas a partir de una o más cámaras. Estos métodos realizan los cálculos de un conjunto de imágenes de la escena y el principio básico utilizado es la triangulación. La visión pasiva presenta ciertas ventajas sobre la visión activa, a menudo es más rápida y menos costosa computacionalmente. Por otra parte, tiene algunas debilidades, tales como la sensibilidad a la transparencia, el riesgo de confusión que puede ser causada por los efectos especulares o inter-reflexiones. Por último, la existencia de texturas sobre los objetos de la escena es considerada como algo positivo para algunos métodos pasivos pero es negativo para otros.

Algunos enfoques pasivos utilizan una sola imagen como la Shape-From-Shading (SFS) **[5]**; procede a la determinación de la forma 3D a través de una variación gradual de la iluminación, lo que permite estimar las normales a la superficie.

Otros enfoques exploran varias imágenes tomadas al mismo tiempo desde diferentes ángulos de la escena, lo que permite desarrollar criterios para la correspondencia entre diferentes puntos de vista para determinar la profundidad de cada punto. Entre estos métodos pasivos multioculares, también se puede citar el Shape-From-Texture (SFT) **[6]** basado en el cálculo de texturas distorsionadas de las superficies de los objetos de la escena. Otro método es también el Shape-From-Motion (SFM) **[7]** se opera, como su propio nombre indica, el movimiento relativo entre la cámara y la escena.

Por último, el método pasivo más conocido es la visión estereoscópica. Precisamente en este proyecto se va a trabajar con la visión estereoscópica por lo que en el siguiente apartado se presentará este método de una manera resumida así como sus diferentes fases.

2.2 Visión estereoscópica

La estereoscopía es un método ampliamente utilizado para reconstruir una escena 3D y extraer la forma de los objetos que contiene. Se trata de la deducción del relieve de los objetos a partir de la diferencia entre las imágenes (normalmente dos) tomadas desde diferentes puntos de vista. Este relieve es simplemente la distancia de los objetos de la escena a la cámara también llamado profundidad.

El más clásico de los métodos de estereoscopía es el estéreo binocular que utiliza dos imágenes de una escena vistas desde dos ángulos ligeramente diferentes, al igual que la visión humana.

Para estimar la profundidad, la geometría relativa de las cámaras debe ser conocida de antemano o ser determinada en una fase de calibración. El procesamiento estéreo por disparidad binocular es tema de mucha investigación desde la aparición de la visión por computador con los trabajos de Marr y Poggio en 1976 **[8]** y 1979 **[9]**, hasta publicaciones recientes de Sanchez-Riera, Cech y Horaud en 2012 **[10]**.

La estéreo ha conocido una expansión de trabajos en los últimos años, motivado por muchas aplicaciones, especialmente en la robótica y las interfaces hombre-máquina (IHM).

El proceso de la estereoscopía es a menudo descrito en tres etapas:

- Pretratamiento
    - Calibración
    - Adquisición
    - Rectificación
- Matching (emparejamiento)
- Reconstrucción (triangulación)

2.2.1 Pre-tratamiento

El pre-tratamiento comienza con una fase de **calibración** (offline) que consiste en encontrar la geometría interna y externa del sistema de adquisición. La calibración interna tiene como objeto encontrar los parámetros intrínsecos de cada cámara tales como la distancia focal, el centro óptico, las dimensiones en píxeles, etc.

La calibración externa, por su parte, tiene como objetivo recuperar la transformación (parámetros extrínsecos) que devuelve la posición de la cámara en una referencia externa (referencia mundo). La fase de calibración se hace a menudo a través de un tablero de ajedrez o un patrón de calibración. Después de la calibración y la **adquisición** de imágenes, es necesario realizar una **rectificación**. Esta rectificación consiste en modificar las imágenes originales para ponerlas en geometría epipolar para después sacar el mapa de disparidad. Se cita el siguiente artículo **[11]**, donde se analiza los avances en visión estéreo.

2.2.2 Matching (emparejamiento)

La etapa matching o emparejamiento consiste en poner en correspondencia los elementos de dos imágenes. Es decir, encontrar los puntos 2D (pixeles) de cada imagen que representan las proyecciones de un mismo punto 3D de la escena (vértice). Varios métodos de puesta en correspondencia (matching) están propuestas en la literatura como la correlación o la búsqueda de puntos de interés, etc.

La puesta en correspondencia de imágenes presenta varias dificultades, tales como el fenómeno de la oclusión o ausencia de texturas. Varios artículos resumen el estado del arte del matching (juego), tales como Scharstein y Szeiliski **[12]** y Brown **[11].**

2.2.3 Reconstrucción

La reconstrucción 3D es la etapa final de la técnica estéreo, que permite estimar la estructura 3D de la escena observada basándose en el principio de triangulación geométrica. Esto consiste en calcular las coordenadas 3D (X, Y, Z) a partir de coordenadas 2D primitivas.

Anteriormente, se ha visto conceptos básicos de la geometría epipolar y otros términos relacionados. También se ha visto que si se tienen dos imágenes de una misma escena, se puede obtener información de la profundidad de una forma intuitiva. A continuación se muestra una imagen y algunas fórmulas matemáticas simples que demuestran esa intuición:

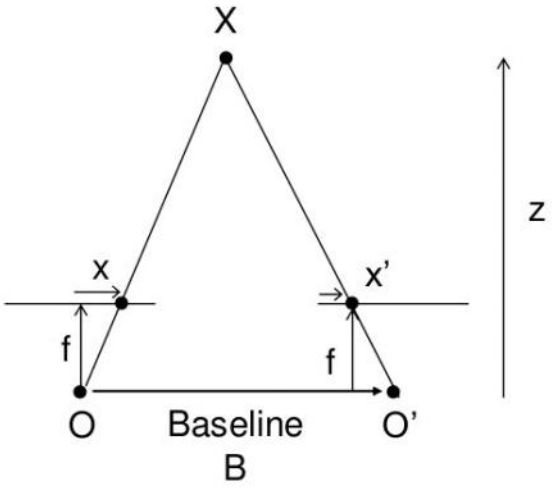


Figura 2: Reconstrucción 3D imágenes estéreo. ***[13]***

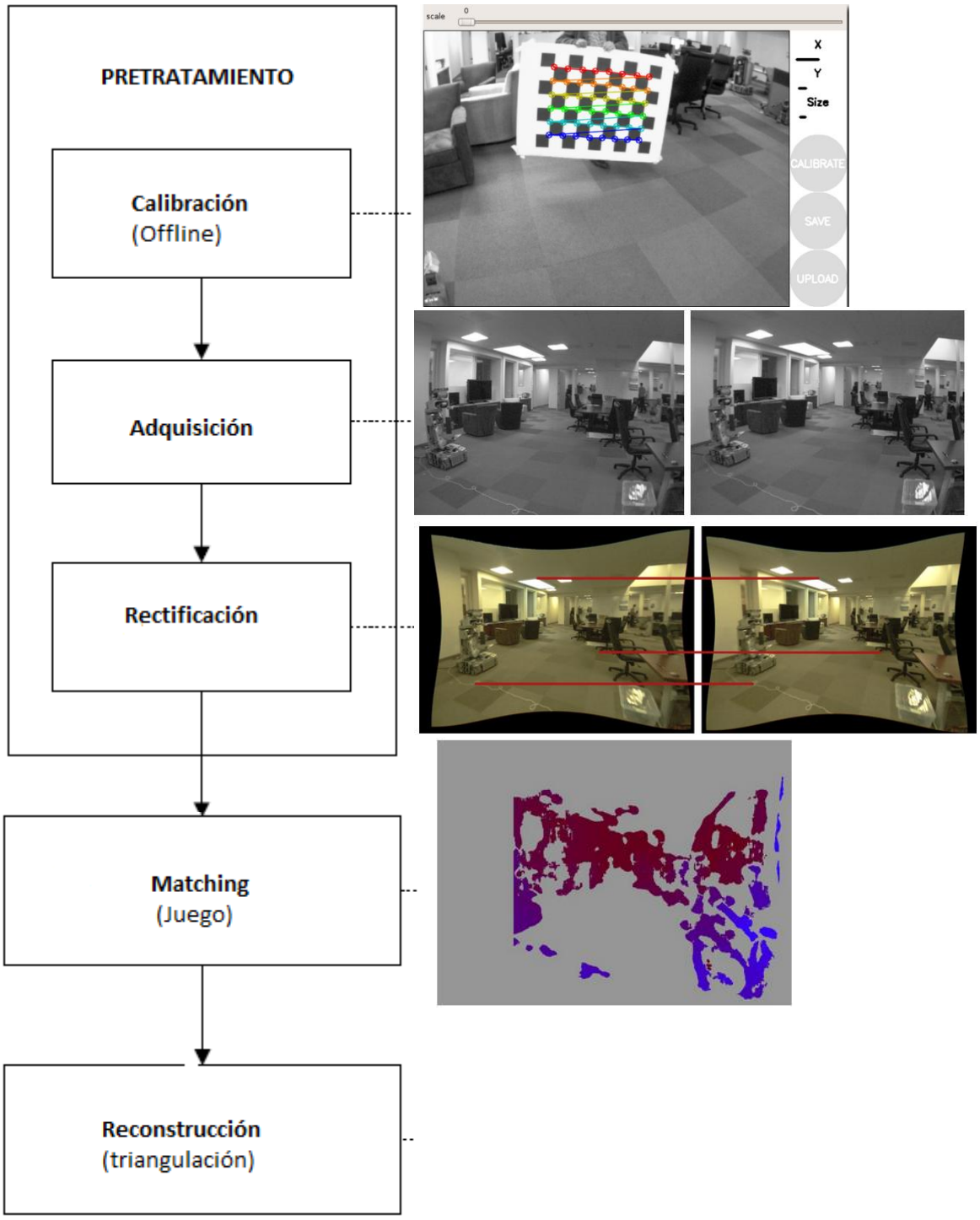
El diagrama anterior contiene triángulos equivalentes. Escribir sus ecuaciones equivalentes proporciona el siguiente resultado:

$$\text{disparity} = x - x' = \frac{Bf}{Z}$$

x y x' son la distancia entre los puntos en el plano de imagen correspondiente a la escena 3D y su centro de la cámara. B es la distancia entre dos cámaras (que se conoce) y f es la distancia focal de la cámara (ya conocida). Así que en resumen, la profundidad de un punto en una escena es inversamente proporcional a la diferencia en la distancia de puntos de imagen correspondientes y sus centros de la cámara. Así que con esta información, se puede deducir la profundidad de todos los píxeles de una imagen.

El siguiente diagrama de bloques resume las diferentes etapas de la visión estéreo:

Figura 3: Diagrama de bloques visión estereo. ***[14]***



2.3 Cámara bumblebee xb3

En este trabajo, se utiliza la cámara estéreo Bumblebee xb3 fabricada por la empresa Point Grey.

"Los sistemas de cámara IEEE-1394 (FireWire) de visión estéreo de Point Grey Research se ofrecen como paquetes de hardware y software completos. Cada sistema de cámaras de visión estéreo de Point Grey incluye una copia gratuita del SDK FlyCapture, que se utiliza para la adquisición de imágenes y control de la cámara, y el Triclops SDK, que lleva a cabo la rectificación de imágenes y procesamiento estéreo." **[15]**

Figura 4: Cámara bumblebee xb3 de point grey. **[15]**

La Bumblebee XB3 es una cámara de 3 sensores de múltiples línea base de IEEE-1394b (800 Mb / s). Es una cámara estéreo diseñada para mejorar la flexibilidad y la precisión. Cuenta con sensores de 1.3 mega-pixel y tiene dos líneas base para el procesamiento estéreo. La línea base larga genera el mapa de disparidad entre las cámaras mas separadas y proporciona una mayor precisión a distancias más largas, mientras que la línea base corta que utiliza las cámaras más cercanas mejora la puesta en correspondencia de la gama cercana y las limitaciones de mínimo alcance.

2.3.1 Componentes y especificaciones técnicas

La bumblebee xb3 se compone de tres fotosensores acopladas del tipo CCD (Charged-Coupled Device), cada uno con una abertura horizontal (HFOV) de 66 grados y separadas de una línea de base de 12 cm y 24 cm de longitud.

La bumblebee xb3 puede recuperar imágenes de resolución 1280x960 a 15 Hz, mientras que si la resolución es más baja por ejemplo 320 x 240 se puede recuperar las imágenes a mayor velocidad.

MODEL	VERSION	FOCAL LENGTH/FOV	MP	IMAGING SENSOR
BBX3-13S2C-38	Color	3.8 mm, 66-deg HFOV	1.3 MP	• Sony ICX445, 1/3", 3.75 µm • Global Shutter • 1280 x 960 at 16 FPS
BBX3-13S2M-38	Mono			
BBX3-13S2C-60	Color	6 mm, 43-deg HFOV		
BBX3-13S2M-60	Mono			

A/D Converter	12-bit
Image Data Output	8, 12, 16 and 24-bit digital data
Image Processing	None
Gain	Automatic/manual/one-push modes
White Balance	Automatic/manual/one-push modes
Color Processing	No on-camera color processing, on-PC in Raw format only
Digital Interface	2 x 9-pin IEEE-1394b for camera control and video data transmit
Transfer Rates	400 Mbps
GPIO	4 general-purpose digital input/output (GPIO) pins
External Trigger Modes	IIDC Trigger Modes 0, 1, 3, and 14
Synchronization	Via external trigger or software trigger
Shutter	Global shutter Automatic/Manual 0.03 ms to 66.63 ms at 15 FPS
Flash Memory	512 KB
Dimensions	277 x 37 x 41.8 mm
Mass	505 grams
Power Consumption	4 W at 12 V via GPIO or FireWire interface
Camera Specification	IIDC v1.31
Camera Control	Via FlyCapture SDK, CSRs, or third party software
Camera Updates	In-field firmware updates
Baseline	12 cm and 24 cm
Aperture	f/2.0 (3.8 mm focal length), f/2.5 (6.0 mm focal length)
Field of View	3.8 mm with 66° HFOV / 6 mm with 43° HFOV
Lens Mount	3 x M12 microlens mount
Temperature	Operating: 0° to 45°C; Storage: -30° to 60°C
Emissions Compliance	CE, FCC, RoHS
Warranty	3 years

Tabla 2: Especificaciónes técnicas cámara Bumblebee xb3. ***[15]***

2.3.2 SDK de visión estéreo TRICLOPS

La bumblebee xb3 viene con paquetes de software: FlyCapture SDK y Triclops SDK. El FlyCapture SDK se utiliza para controlar los parámetros de la cámara estéreo y para la adquisición de imágenes. El Triclops SDK se utiliza para proporcionar y estimar la profundidad de cada pixel en tiempo real a partir de las imágenes mediante algoritmos de estereoscopía, es decir, rectificación de imágenes y procesamiento estéreo. De hecho, Triclops SDK permite a los usuarios medir con precisión la distancia a cada píxel válido en la imagen.

Ambos SDKs incluyen drivers para cámaras, librerías completas y la interfaz de programación de aplicaciones (API) para su uso en el entorno de programación C / C ++. La cámara se conecta al ordenador a través de una interfaz FireWire IEEE-1394 (véase el apartado anterior para las especificaciones técnicas de la bumblebee xb3).

2.4 ROS

2.4.1 Introducción a ROS

ROS es un meta-sistema operativo, de código abierto para robots basado en bloques que comparten mensajes y utilizan herramientas en común con capacidad de depuración y reusabilidad.

ROS permite la implementación de código de uso común, paso de mensajes entre procesos y gestión de paquetes. También proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en los equipos.

En este trabajo se va a utilizar principalmente para escribir y ejecutar códigos para el paso de mensajes entre procesos (nodos).

Figura 5: Logotipo ROS. ***[16]***

Para la instalación de este programa existe un instalador debian que gestiona todas las dependencias que va a necesitar, dependiendo de la versión que se quiera instalar.
ROS está bajo licencia de código abierto y todos los pasos para la instalación se pueden encontrar en **[16].**

Una vez instalado ROS, es importante crear y configurar el "workspace" (que es el sitio donde se crearán los paquetes para trabajar en ROS) ya que se creará una variable de entorno de esta carpeta para que ROS sepa dónde buscar los paquetes que se van a usar.

2.4.2 Objetivos de ROS

En general el objetivo principal de ROS es unificar la comunidad robótica y estandarizar de forma abierta y actualizable, las aplicaciones de bajo nivel que sean básicas, para focalizar esfuerzos en objetivos de mayor nivel.

Siendo su filosofía crear un sistema operativo que reutilice el código que se ha desarrollado.

2.4.3 Sistemas operativos

El software de ROS ha sido probado principalmente en Ubuntu y Mac OS X, aunque también se ha probado en otras plataformas.

La utilización de ROS en Microsoft Windows es posible, pero todavía no se ha explorado a fondo.

En este trabajo se ha utilizado tanto la versión de unbuntu 11.10 como la 12.04.

2.4.4 Versiones de ROS

Las herramientas y librerías de ROS son distribuidas periódicamente como una versión de ROS.

Las últimas versiones de ROS son las siguientes:

Distro	Release **date**	Poster	*Tuturtle*, turtle in tutorial
ROS Indigo Igloo (**Recommended**)	July 22nd, 2014		
ROS Hydro Medusa	September 4th, 2013		
ROS Groovy Galapagos	December 31, 2012		

Tabla 3: Últimas versiones de Ros. **[16]**

Para la realización del trabajo se ha empezado con ROS fuerte, después se ha migrado a la versión ROS hydro, debido a las actualizaciones que se han realizado en el coche IVVI 2.0.

2.4.5 El sistema de archivos de ROS

Los recursos de ROS más utilizados en el trabajo están organizados en:
Paquetes: Un paquete puede contener procesos de ejecución (nodos), conjuntos de datos, archivos de configuración, o cualquier otra cosa que sea útil.

Manifest.xml: proporcionan metadatos sobre un paquete, incluyendo su información de licencia y dependencias, así como información específica del idioma.
Mensajes (msg); los tipos de mensajes: definen las estructuras de datos para los mensajes enviados en ROS.

2.4.6 ROS a nivel de ejecución

Los conceptos más importantes de ROS y utilizados en este trabajo son nodos, mensajes, topics (temas) y bags (bolsas), los cuales proporcionan los datos de ejecución de diferentes maneras.

Nodos: Los nodos son procesos, por ejemplo, un nodo publica las imágenes de una cámara, un nodo proporciona una vista gráfica del sistema, etc.

Mensajes: Los nodos se comunican entre sí pasando mensajes. Un mensaje es simplemente una estructura de datos.

Topics (Temas): Los mensajes se enrutan a través de un sistema de transporte de publicación/suscripción semántica. Un nodo envía un mensaje mediante su publicación a un determinado tema. El tema es un nombre que se utiliza para identificar el contenido del mensaje.

Bags (Bolsas): Las bolsas son un formato para guardar y reproducir datos de mensajes de ROS. Esto sirve por ejemplo para recoger imágenes en el laboratorio y guardarlas en unos bagfiles para después trabajar con estas imágenes en otro ordenador.

A continuación se demuestra una imagen donde se ilustra ros a nivel de ejecución:

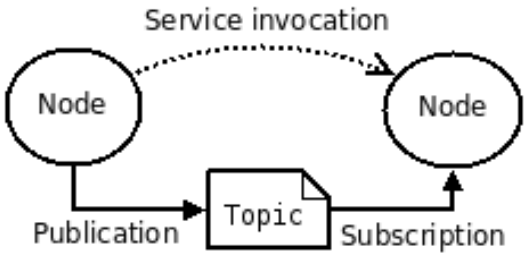


Figura 6: Ros a nivel de ejecución. **[16]**

2.5 OpenCV en ROS

Figura 7: Logtipo OpenCV **[17]**

Al instalar ROS, directamente se instala también la biblioteca OpenCV (aunque en la versión ROS Hydro no existe un paquete de OpenCV en ROS sino la versión standalone de OpenCV2, es decir funciona de forma independiente) por lo tanto se puede conectar ROS y OpenCV mediante la conversión de imágenes de ROS en imágenes OpenCV, y viceversa, utilizando cv_bridge. También se tiene instalado OpenCV independiente de ROS ya que se va a necesitar a lo largo del desarrollo del proyecto.

"*La biblioteca OpenCV es una biblioteca gráfica libre bajo licencia BSD y tiene más de 500 funciones muy utilizadas en la visión artificial, procesamiento de imágenes, detección de movimiento, reconocimiento de objetos etc. Además es multiplataforma existiendo versiones para Linux, Mac OS X y Windows*." **[17]**

Cuenta con interfaces de C + +, C, Python y Java y fue diseñado para la eficiencia computacional y con un fuerte enfoque en las aplicaciones en tiempo real. Programada en C / C + +, esta biblioteca puede tomar las ventajas del procesamiento multi-core.

Al instalar OpencCV se crean los siguientes directorios:

Bin. Archivos DLL (Dynamic Link Library), y algunos ejecutables de prueba, test e información de las librerías.

Imgproc. (Image Processing). Módulo que incluye algoritmos para el procesamiento de imágenes.

cxcore. Código fuente del núcleo de la librería, operaciones de bajo nivel sobre arrays, matrices e imágenes.

Core. (The Core functionnalities). Módulo que define las estructuras de datos y funciones básicas empleadas en el resto de los módulos.

Video. (Video analysis). Módulo que incluye algoritmos para el análisis de videos.

Cv. Código fuente de la librería, operaciones de procesamiento de imágenes y visión artificial.

highgui. Librerías para crear ventanas, leer y escribir imágenes (formatos BMP, JPEG, PNG y TIF), archivos de vídeo (formatos AVI, MPG, WMV y MOV) y captura de cámara.

opencv. Contiene los ficheros de cabecera necesarios para incluir en los programas C/C++. Los importantes son: cxcore.h, cv.h y highgui.h.

Calib3d. Módulo con algoritmos para deducir las características geométricas de un espacio a partir de múltiples vistas.

Features2d. (Feature detection and descriptor extraction).Módulo con algoritmos de cálculo de descriptores y características.

Calib3d. (Camera calibration, pose estimation and stereo).Módulo con algoritmos para deducir las características geométricas de un espacio a partir de múltiples vistas.

doc . Documentación de las librerías. La parte principal de la documentación (el manual de referencia de OpenCV) está en el fichero **opencv.pdf**.

lib. Ficheros de descripción de las librerías

samples\c. Programas sencillos en C que muestran diversas funciones de OpenCV.

Y Otros directorios, pero se puede decir que los mencionados anteriormente son los más importantes sobre todo para el caso de este trabajo.

ROS pasa las imágenes en su propio formato de mensaje *sensor_msgs / Image* , pero en muchos casos se quiere utilizar imágenes en conjunto con OpenCV. CvBridge en un paquete de ROS que proporciona una interfaz entre ROS y OpenCV. CvBridge se puede encontrar en el paquete cv_bridge en el stack vision_opencv.

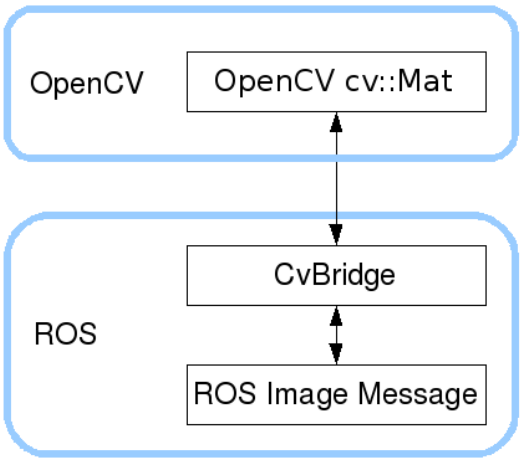


Figura 8: cv_bridge ***[18]***

Como se puede observar en la figura anterior OpenCV almacena los datos de las imágenes en la estructura Mat y dispone de potentes algoritmos para trabajar sobre imágenes.

2.6 Lenguaje de programación

El lenguaje de programación utilizado en este trabajo es C++. Este lenguaje de programación es uno de los más populares, con una amplia variedad de plataformas de hardware y sistemas operativos. El marco ROS es fácil de implementar en cualquier lenguaje de programación moderno, C++, Python; e incluso actualmente existen bibliotecas experimentales en Java y Lua.

C ++ es un lenguaje de programación que permite la programación bajo múltiples paradigmas como la programación estructurada, orientada a objetos y la programación genérica. C ++ no pertenece a nadie, por lo que cualquier persona puede utilizarlo sin necesidad de autorización u obligación de pagar por el derecho de uso.

Por lo tanto se ha realizado este trabajo en lenguaje de programación C++, utilizando Ros bajo Ubuntu, se ha utilizado tanto la versión Ros fuerte como la Hydro y también se ha usado Ubuntu 11.10 y 12.04.

3 Generación de mapas de disparidad

La reconstrucción de la información 3D a partir de pares de imágenes estéreo es uno de los principales problemas en la visión artificial y análisis de imagen. Se han propuesto una variedad de algoritmos con este propósito para calcular un mapa de disparidad utilizando las imágenes estéreo. Un mapa de disparidad es una imagen de la que los datos 3D se pueden extraer para cada píxel utilizando los parámetros de calibración. Básicamente, el mapa de disparidad es la diferencia en la ubicación de la imagen de cada píxel visto por la cámara izquierda y la derecha y se calcula mediante el establecimiento de correspondencias entre estos píxeles.

3.1 Mapas de disparidad con OPenCV en ROS

En este capítulo se va a comprobar la rectificación de las imágenes con el código que ya viene programado en el driver xb3 utilizando OpenCV y ROS. En este driver se utiliza la biblioteca OpenCV para hacer la rectificación, se almacenan las imágenes en matrices Cv Mat de OpenCV y se convierten en mensajes ROS utilizando cv_bridge, un módulo de ROS que es capaz de convertir imágenes OpenCV a mensajes ROS y viceversa.

Por lo tanto los objetivos a desarrollar en este capítulo son:

- Publicar las imágenes de la cámara bumblebee mediante el driver xb3 a ROS.
- Comprobar la rectificación de las imágenes, inspeccionándolas.
- Realizar disparidad “wide” y “narrow” mediante distintos códigos y ajustando los parámetros.

Disparidad Narrow: disparidad obtenida a partir de las imágenes de dos de las cámaras más cercanas, sea right y middle o middle y left.

Disparidad Wide: disparidad obtenida a partir de las imágenes de las dos cámaras más lejanas, en este caso, right y left.

3.1.1 Publicación de imágenes mediante el driver xb3 a ROS

Utilizando el driver xb3, se publican las imágenes a ROS con los siguientes topics:

```
abdoulaye@ubuntu:~/bagfiles$ rostopic list
/clock
/rosout
/rosout_agg
/stereo_camera/image_raw
/xb3/left
/xb3/merged
/xb3/middle
/xb3/right
```

Figura 9: rostopic list

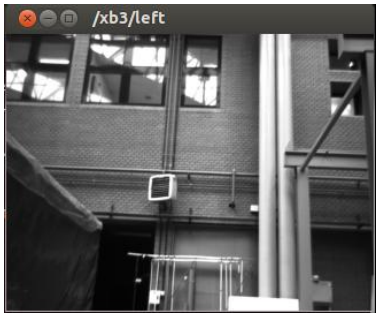
Al principio se extrae de la cámara tres imágenes que llegan al ordenador como una sola imagen RGB unidas por el tipo de bits y se utiliza el driver xb3 para separar esta imagen en tres imágenes. En la figura 9 se puede observar que se están publicando 5 imágenes, image_raw (imagen sin desentrelazar), merged (las tres imágenes mezcladas), la imagen derecha de la cámara, centro e izquierda respectivamente.

a) Image_raw

b) xb3_merged

c) Right_image

d) Middle_image



e) Left_image

Figura 10: Imágenes Bumblebee xb3

A continuación se presenta un ejemplo del listado de todos los topics publicados con la cámara en pleno funcionamiento:

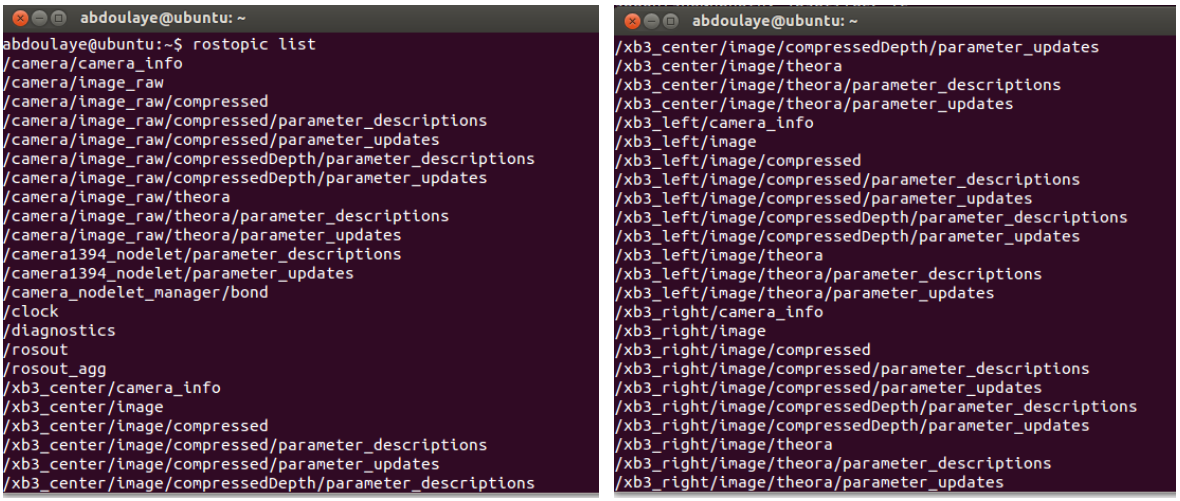


Figura 11: Topics de la cámara

Utilizando rxgraph de ROS se puede observar cómo están conectados todos los nodos y la comunicación que hay entre ellos, es decir, se puede ver que topics se suscriben y/o se publican en cada nodo:

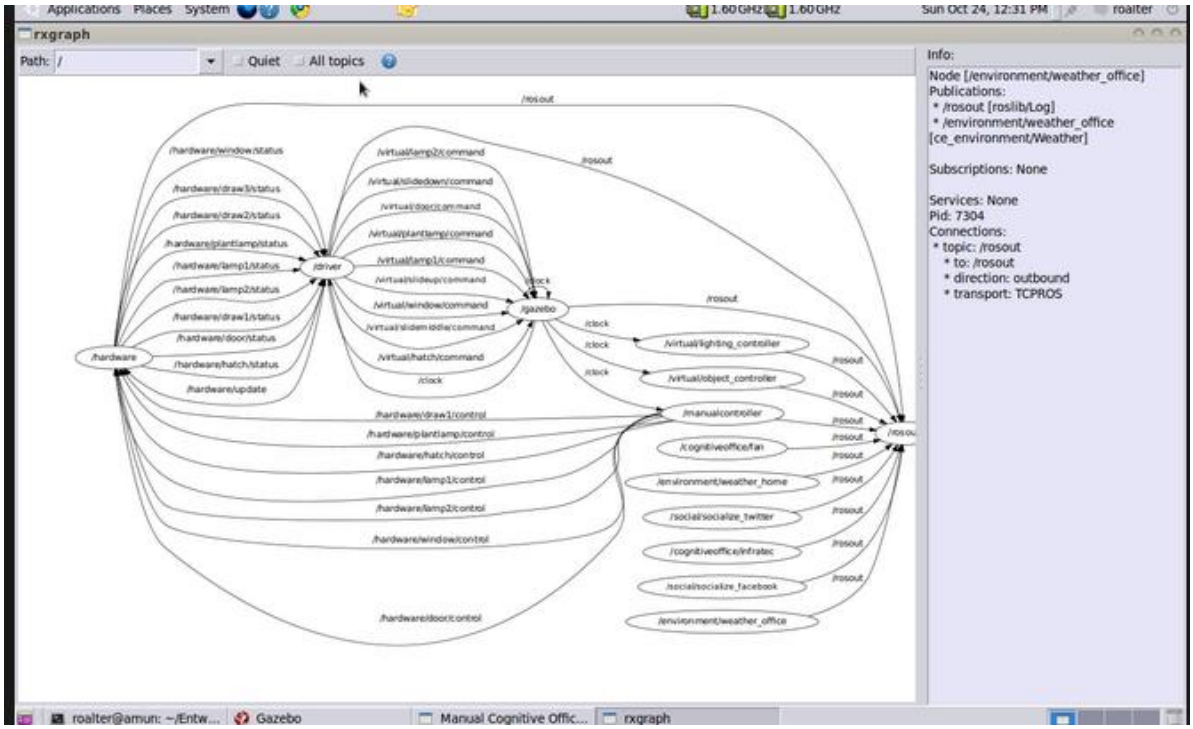
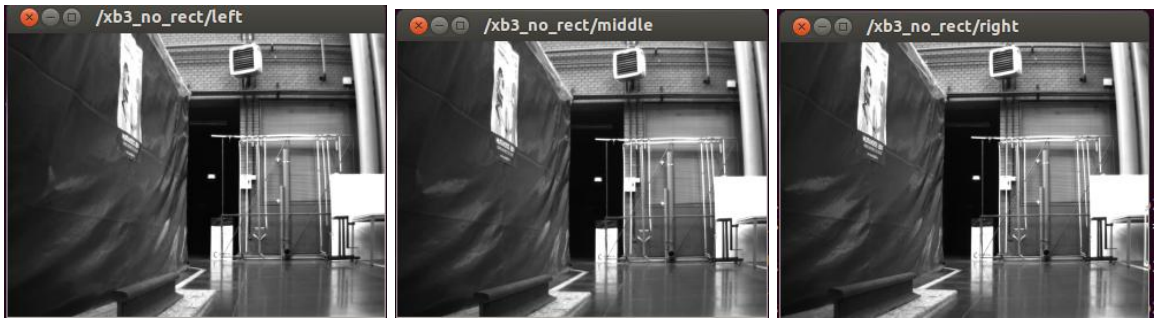


Figura 12: Conexión de nodos en ROS

Se modifica el driver xb3 para poder observar también las imágenes no rectificadas, right, middle y left y así comparar también el resultado visualmente:

```
abdoulaye@ubuntu:~/stereo$ rostopic list
/clock
/rectifier_xb3/left
/rectifier_xb3/merged
/rectifier_xb3/middle
/rectifier_xb3/right
/rosout
/rosout_agg
/xb3/camera_info
/xb3/image_raw
/xb3_no_rect/left
/xb3_no_rect/merged
/xb3_no_rect/middle
/xb3_no_rect/right
```

Figura 13: Lista de topics sin rectificar



a) Unrectified_left *b) Unrectified_middle* *c) Unrectified_right*

d) rectified_right *e) rectified_middle* *f) rectified_left*

Figura 14: Imágenes con driver xb3 modificado.

En las imágenes de la figura 14 a), b) y c) se observa claramente que las imágenes no están rectificadas, los objetos están torcidos y la diferencia con las imágenes d), e) y f) del mismo entorno es bastante apreciable.

3.1.2 Comprobación de la rectificación de las imágenes

Para comprobar la correcta rectificación de las imágenes, se sacan los mapas de disparidad utilizando varios códigos y se analizan estos mapas. Un mapa de disparidad es una colección de distancias de correspondencia entre los pixeles de dos imágenes estudiadas. A continuación se presenta un ejemplo de mapa de disparidad con dos imágenes de entrada que se muestran en la figura 15 a) y b). Estas dos imágenes son capturadas por una cámara estéreo con sus cámaras izquierda y derecha respectivamente, la imagen c) de la misma figura es el mapa de disparidad, que es la colección de distancia entre cada pixel de la imagen izquierda y el correspondiente pixel de la imagen de la derecha. Visualmente, el objeto más próximo tendrá una disparidad más elevada.

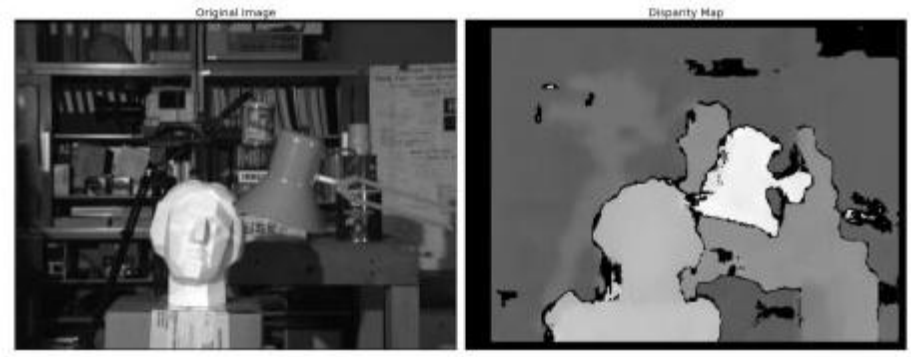
a) Rectified_left_image *b) Rectified_right_image* *c) Correct_disparity*

Figura 15: Imágenes rectificadas con disparidad correcta ***[19]***

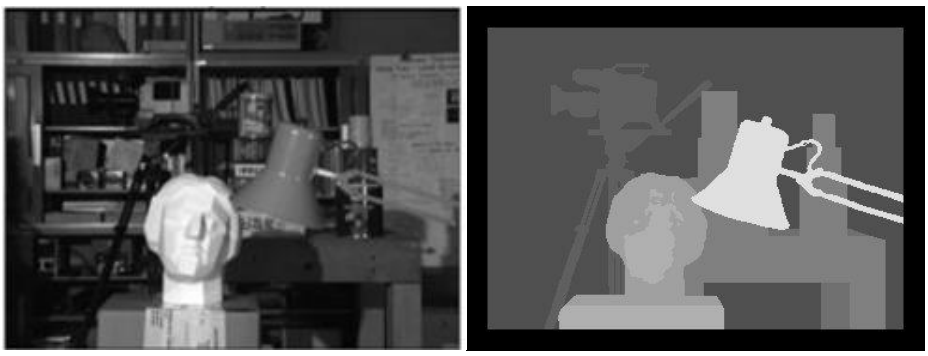
El mapa de disparidad está directamente relacionado con el mapa de profundidad. Se Puede crear un mapa de profundidad a partir de un mapa de disparidad si se conoce la longitud focal de la cámara.

Para comprobar la rectificación de imágenes mediante los mapas de disparidad, hay que tener en cuenta que hay que ajustar los parámetros del código de disparidad porque de lo contrario se podría obtener un mapa erróneo, pero no porque las imágenes estén mal rectificadas sino porque no se han ajustado bien los parámetros.

Por ejemplo la figura 16 contiene la imagen original a) y su mapa de disparidad b). El resultado está contaminado con alto grado de ruido. Mediante el ajuste de determinados parámetros, se obtiene mejor resultado en figura 16 d).



a) Original_image *b) Incorrect_disparity*



c) Original_image *d) Correct_disparity*

Figura 16: Ilustración de correcta disparidad ***[13]***

3.1.3 Disparidad Narrow y Wide

Para realizar la disparidad de las imágenes se utiliza el Block matching de OpenCV, aprovechando la compatibilidad de ROS y OpenCV.

Block matching es el método más utilizado para la estimación de la disparidad y es simple, eficaz y fácil de implementar. La idea básica de block matching es segmentar la imagen de destino en tamaño fijo de bloques y encontrar para cada bloque su correspondiente que ofrece la mejor coincidencia en la imagen de referencia.
En general, el bloque que minimice la estimación de error es normalmente seleccionado como el bloque correspondiente. De todas formas, block matching a veces no es preciso y no da mapas de disparidad precisos pero es muy utilizado para sacar mapas de disparidad de imágenes estéreo. Otros métodos que se podrían utilizar y quizás sean más fiables son el método Semi-Global Matching (SGM) **[20]** y the Graph-Cut-Based exist **[21]** pero necesitan más potencia de procesamiento para ser utilizado en tiempo real y producen mapas de disparidad densos.

A continuación se presentan los parámetros del block matching de OpenCV:

StereoBM()

C++: StereoBM::StereoBM()

C++: StereoBM::StereoBM(int **preset**, int **ndisparities**=0, int **SADWindowSize**=21)

Parámetros:

- **Preset -**

  Especifica el conjunto de parámetros del algoritmo, estos son:

  - BASIC_PRESET - Parámetros adecuados para cámaras generales
  - FISH_EYE_PRESET - Parámetros adecuados para cámaras de ángulo ancho.
  - NARROW_PRESET – Parámetros adecuados para cámaras de ángulo estrecho.

- **ndisparities** – El rango de búsqueda de disparidad. Para cada pixel, el algoritmo encontrará la mejor disparidad de 0 (mínima disparidad por defecto predeterminado) a ndisparities. El rango de búsqueda puede entonces ser desplazado cambiando la disparidad mínima.

- **SADWindowSize** – El tamaño lineal de los bloques comparado por el algoritmo. El tamaño debe ser impar (ya que el bloque está centrado en el píxel actual). Mayor tamaño de bloque implica mayor suavidad, aunque menos precisión en el mapa de disparidad. Menor tamaño de bloque da un mapa de disparidad más detallado, pero hay una probabilidad muy alta de que el algoritmo encuentre una correspondencia errónea.

A continuación, se presenta el resultado de los mapas de disparidad narrow y wide obtenidos a partir de las tres imágenes:

a) xb3_right. *b) xb3_middle* *c) xb3_left.*

Figura 17: Imágenes rectificadas con xb3 sin modificar.

Figura 18: Narrow disparity.

Figura 19: Wide disparity.

Como se puede observar en las imágenes anteriores, el resultado obtenido es bastante malo, ya que este resultado presentado aquí es el mejor que se ha obtenido en este caso tras varios otros bastante peores y éste se ha obtenido después de haber ajustado todos los parámetros.

Para solventar el problema, se procede a realizar algunas modificaciones en el código del driver xb3, intercambiando los parámetros de rectificación asignados a las cámaras izquierda y derecha. Esto se hace para verificar que el driver xb3 original está bien programado y no se han asignado los parámetros de la cámara derecha a la izquierda y viceversa por error.

Como se puede apreciar en las imágenes de la figura 20 y 21, el resultado no ha sido satisfactorio porque incluso ha empeorado bastante.

Figura 20: Modified narrow_disparity

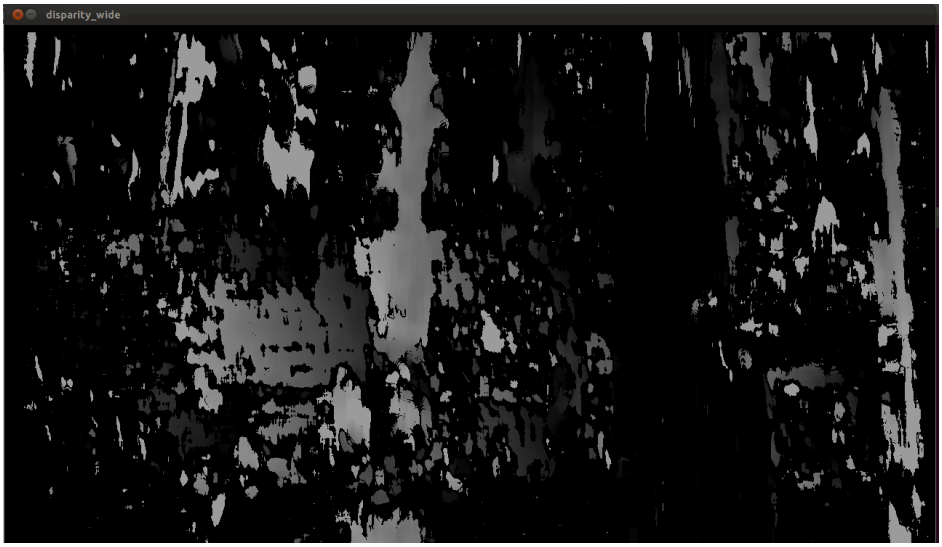


Figura 21: Modified Wide_disparity

Después de los malos resultados anteriores, se decide cambiar el código con el que se está realizando la disparidad, utilizando el mismo método (stereoBM) pero ahora con un nuevo código que permite ajustar más parámetros. A continuación se presentan los parámetros importantes del nuevo código:

StereoBM

```
StereoBM sbm;
sbm.state->SADWindowSize = 9;
sbm.state->numberOfDisparities = 112;
sbm.state->preFilterSize = 5;
sbm.state->preFilterCap = 61;
sbm.state->minDisparity = -39;
sbm.state->textureThreshold = 507;
sbm.state->uniquenessRatio = 0;
sbm.state->speckleWindowSize = 0;
sbm.state->speckleRange = 8;
sbm.state->disp12MaxDiff = 1;
```

- **minDisparity** – Mínimo valor de disparidad posible.
- **numDisparities** – Disparidad máxima menos disparidad mínima. Este parámetro ha de ser divisible entre 16.
- **SADWindowSize** –Tamaño de bloque emparejado. Debe ser un número impar> = 1.
- **disp12MaxDiff** – Máxima diferencia permitida (en unidades de píxel enteros) en el chequeo de disparidad entre izquierda y derecha.
- **preFilterCap** – Valor de truncamiento de los píxeles de la imagen pre filtrados de la imagen.
- **uniquenessRatio** – Margen de porcentaje en el que el mejor (mínimo) valor de función de coste calculado debe "ganar" el segundo mejor valor para considerar la correspondencia encontrada correcta. Normalmente, un valor dentro del rango de 5-15 es bastante bueno.
- **speckleWindowSize** – Tamaño máximo de las regiones de disparidad suaves para considerar sus motas de ruido e invalidar.
- **speckleRange** – Variación máxima de disparidad dentro de cada componente conectado.

Con los parámetros anteriores bien ajustados ya se puede calcular disparidad. Como la disparidad será o bien CV_16S ó CV_32F, necesita ser comprimido y normalizado a CV_8U.

Con las imágenes guardadas en unos bagfiles de ROS, se extraen para poder tenerlas en formato foto.jpg y realizar la disparidad con el nuevo código, obteniendo los siguientes resultados tanto con el driver xb3 modificado como sin modificar, después de varios ajustes de los parámetros:

Figura 22: Left_right disparity.

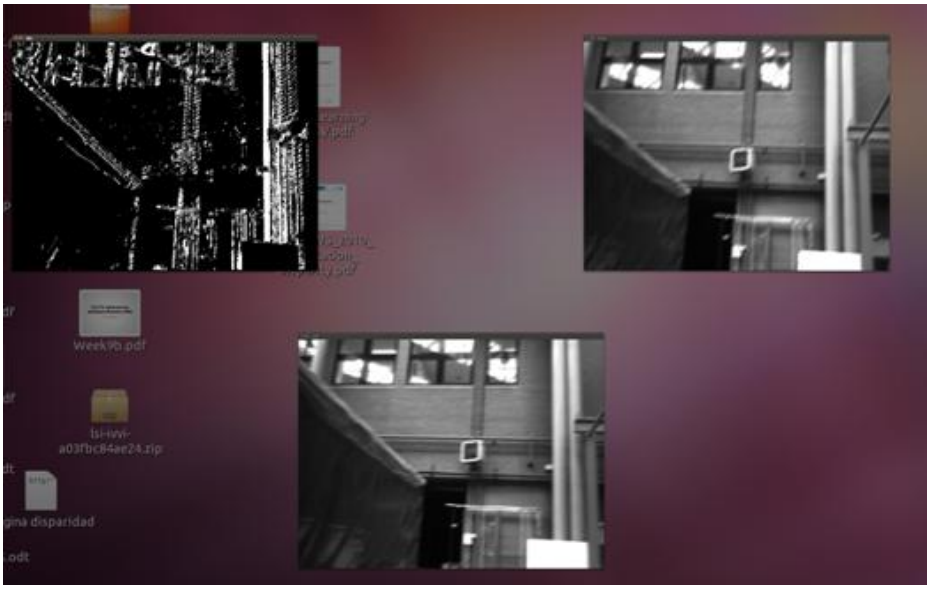


Figura 23: Modified *Left*_right_disparity

El resultado de la disparidad ha mejorado pero se considera que no es un buen resultado, por lo que hay que buscar otra forma de realizar la disparidad, volver a calibrar la cámara o directamente volver a rectificar las imágenes con otro método.

3.1.4 Calibración de la cámara

Después de los malos resultados de la etapa anterior, se considera que será necesario volver a calibrar la cámara. Se procede a realizar una nueva calibración con el package **camera_calibration** de ROS.

Esta calibración se puede realizar de dos formas:

1. Realizar la calibración de las tres cámaras derecha, centro e izquierda por separado con monocular_camera de ROS.
2. Realizar la calibración de las cámaras de dos en dos con stereo_camera de ROS, en este caso se puede realizar una calibración narrow (con las dos cámaras más cercanas) y una calibración wide (con las dos cámaras más lejanas).

Se procede a realizar la calibración de las dos formas mencionadas anteriormente, obteniendo los nuevos parámetros de la cámara que se guardan en unos ficheros que se generan automáticamente al realizar la calibración en ROS. La calibración con stereo_camera se ha realizado de la siguiente manera:

- Stereo_camera, narrow_stereo/left_middle:
- Stereo_camera, narrow_stereo/right_middle:
- Stereo_camera, wide_stereo/right_left:

A pesar de la nueva calibración de la cámara, no se consigue que mejoren los resultados de los mapas de disparidad, es decir, la rectificación de las imágenes y tampoco se puede utilizar estos parámetros generados con ROS en OpenCV porque este último tiene su propia forma de realizar la calibración y genera parámetros distintos. Pero se decide no realizar la calibración con OpenCV ya que éste es menos preciso que ROS y seria simplemente una pérdida de tiempo.

3.1.5 Conclusión

Se llega a la conclusión de que las imágenes están mal rectificadas, pero antes hay que realizar un estudio de las imágenes de disparidad.

Por lo tanto, ahora las nuevas tareas son:

i. Utilizar directamente el programa de pointgrey (triclops) para rectificar y hacer la imagen de disparidad también con triclops. Pero antes hay que instalar este programa en Linux y programar los triclops para que rectifique y saque las tres imágenes.
ii. Con esas mismas imágenes rectificadas, utilizar OpenCV para realizar la imagen de disparidad.
iii. comparar el resultado.
iv. Si el resultado de la rectificación con los triclops es bueno, se realizará una memoria compartida para poder publicar las imágenes rectificadas con los triclops a ROS.

3.2 Mapas de disparidad con driver privativo triclops

En este capítulo se van a presentar los resultados de la rectificación de las imágenes con el software del fabricante. La bumblebee xb3 viene con el software Triclops SDK que se utiliza para la rectificación de las imágenes y el procesamiento estéreo, es decir, para proporcionar y estimar la profundidad de cada pixel en tiempo real a partir de las imágenes mediante algoritmos de estereoscopía.

Básicamente la librería Triclops SDK ha sido desarrollada utilizando un algoritmo de correlación, consiguiendo que el algoritmo se comporte de una forma robusta.

Triclops proporciona un conjunto de funciones de alto nivel para el procesamiento estéreo, cálculo de la disparidad, interpolación sub-pixel, filtrado de disparidad, etc.

La utilización de la bumblebee xb3, con tres cámaras en vez de dos (como por ejemplo la bumblebee xb2) hace posible que la disparidad pueda ser calculada en dos sentidos, wide (izquierda y derecha) y narrow (centro e izquierda). Combinando las dos imágenes de disparidad resultantes se obtiene una imagen de disparidad final de mayor calidad.

Los objetivos de este capítulo son:

- Instalación triclops en Linux.
- Rectificación de imágenes con triclops.
- Disparidad de las imágenes.
- Comparación OpenCV-Triclops.

3.2.1 Instalación de triclops en linux

La cámara estéreo Bumblebee tiene una excelente reputación por su rendimiento en la visión estéreo. Sin embargo, es muy "exigente" en su plataforma de funcionamiento: sólo funciona bajo el sistema Linux de 32 bits debido a la biblioteca estática de 32 bits proporcionada por Point Grey Research.

Algunos prerrequisitos son los siguientes:

Elementos	**No**ta
Sistema Linux de 32 bits	**Ubuntu 12.04**
Ordenador con interfaz ieee1394	**Firewire (1394)**
Modulo kernel instalado	**ieee1394, ohci1394, raw1394**
Librerías instaladas	**libdc1394 (>=2.0.2), libraw1394 (>=1.2.0)**

Tabla 4: Elementos previamente instalados

Point Grey Research, proporciona la Triclops SDK, basada en una correlación por zonas con SAD **[22]** (suma de las diferencias absolutas), para llevar a cabo la rectificación de imágenes y el procesamiento estéreo.

El algoritmo es bastante robusto y tiene un número de etapas de validación que reducen el nivel de ruido. El método requiere la textura y el contraste para que funcione correctamente.

Para instalar esta librería, primero hay que descargar el package: Triclops3.2.0.8-FC3.tgz, registrándose previamente en el sitio web de Point Grey Research.

En segundo lugar, también es necesario descargar el paquete de ejemplos- **pgr-stereo-examples-libdc-2.0.2.tar.gz** que contiene algunos programas de ejemplo y una biblioteca estática libpgrlibdcstereo.a.

Se puede descomprimir el archivo Triclops3.2.0.8-FC3.tgz al directorio / usr / local / include:

~$ sudo tar zxvf Triclops3.2.0.8-FC3.tgz -C /usr/local/include

Después descomprimir el archivo **pgr-stereo-examples-libdc-2.0.2.tar.gz** en otro lugar, por ejemplo */opt*.

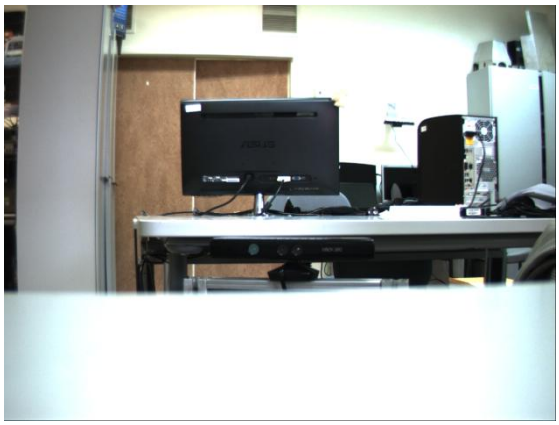
~$ sudo tar zxvf pgr-stereo-examples-libdc-2.0.2.tar.gz -C /opt

Ahora hay que ir al subdirectorio *pgrlibdcstereo,* cambiar el makefile tal y como aparece en **[23],** para generar la librería *pgrlibdcstereo.a.* Una vez generada esta librería hay que enviarla al subdirectorio /lib del archivo triclops. También hay que enviar al subdirectorio

/include de triclops todos los ".h" del archivo **pgr-stereo-examples-libdc-2.0.2.tar.gz** una vez descomprimido.
Ahora ya se puede probar los programas de ejemplo que vienen con el paquete, para ello hay que ir al directorio del programa que se quiera probar y modificar el makefile al igual que se hizo antes, para ver como se hace esto mirar **[23]**. Si, haciendo los pasos anteriores se consigue generar los ejecutables de los programas de ejemplo, entonces los triclops se han instalado correctamente y todo está listo para usar.

3.2.2 Rectificación de imágenes con triclops

Después de instalar los triclops en el sistema operativo (Ubuntu 12.04) y utilizar los programas que vienen como ejemplo, se consigue obtener a partir de la cámara imágenes sin rectificar (figura 24. a y b) así como una imagen perfectamente rectificada (figura 24.c) y su mapa de disparidad (figura 24.d).



a) Unrectified triclops_left.

b) Unrectified triclops_right.

c) Rectified triclops_image.

d) Triclops disparity_image.

Figura 24: Imágenes de triclops.

Con este programa de ejemplo que viene con el software y que se usa aquí, solo se consigue obtener dos imágenes, la de la derecha e izquierda, pero no se consigue la imagen de la cámara del centro, esto es así porque este programa está preparado para la bumblebee xb2, por lo que hay que programar y adaptar el código a la bumblebee xb3 para conseguir las tres imágenes.

Con el código ya programado y adaptado a la bumblebee xb3, se consiguen las tres imágenes perfectamente rectificadas:

a) Triclops_Right_Rectified *b)Ttriclops_Middle_Rectified* *c) Triclops_Left_Rectified*

Figura 25: Las tres imágenes de triclops rectificadas

3.2.3 Disparidad de las imágenes

En este apartado se van a presentar los resultados de los mapas de disparidad obtenidos utilizando los triclops. Los triclops utilizan un método llamado SAD "suma de las diferencias absolutas" para calcular los mapas de disparidad.

El método de correlación SAD es una de las formas más sencillas de establecer correspondencias entre imágenes. La intuición detrás de este enfoque es que hay que hacer lo siguiente:

Para cada píxel en la imagen

- Seleccionar un área de un tamaño cuadrado dado de la imagen de referencia
- Comparar este área a una serie de áreas de la otra imagen (realizando un barrido a lo largo de la misma fila basándose en la línea epipolar).
- Seleccionar la mejor correspondencia.

Los Triclops tienen acceso a las imágenes perfectamente rectificadas y pueden hacer uso de la línea epipolar de una manera más precisa para encontrar las correspondencias.

En la figura 26 se presentan algunos mapas de disparidad generados por los triclops, donde se puede observar que hay una gran diferencia con los obtenidos anteriormente con OpenCV:

a) Triclops_left

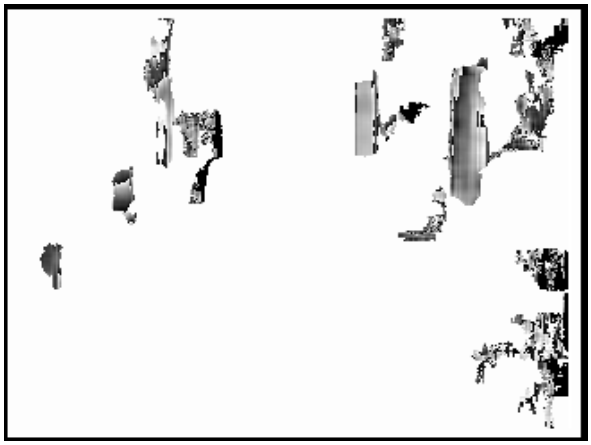
b) Triclops_right

c) Left_disparity

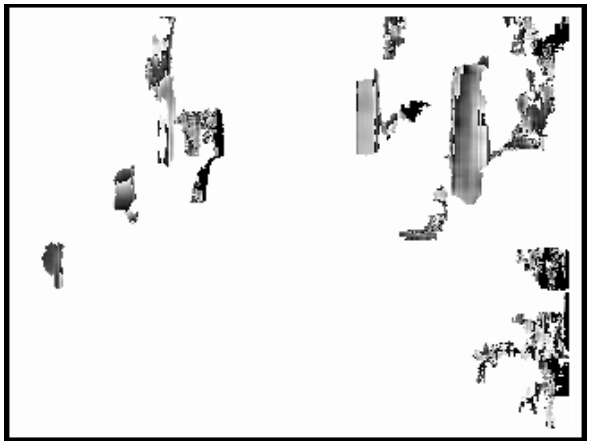
d) Right_disparity

e) Triclops_left

f) Triclops_right



g) Left_disparity



h) Right_disparity

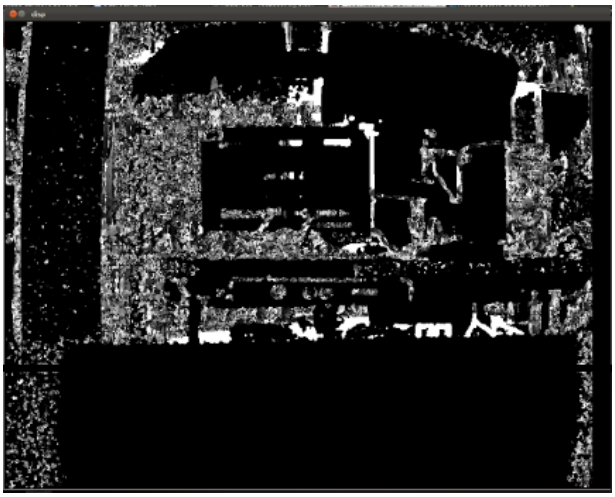
Figura 26: Mapas de disparidad realizados con triclops

3.2.4 Comparación OpenCV vs Triclops

A continuación se presentan dos mapas de disparidad, uno de ellos (figura 27.b) obtenido con los triclops y el otro (figura 27.c) con OpenCV:

a) Triclops rectified_image

b) Triclops_disparity



c) Opencv_disparity

Figura 27: Comparación mapas de disparidad

En los mapas de disparidad anteriores, se puede observar que el generado por los ticlops tiene mucha mayor calidad que el generado por el block maching de OpenCV. Esto es debido a que Triclops tiene acceso directo a las imágenes perfectamente rectificadas y puede hacer uso de la línea epipolar de una manera más precisa para encontrar las correspondencias.

A pesar de que el método Block matching de OpenCV es menos preciso que el algoritmo SAD, se puede utilizar éste en caso de que no se precise de unas imágenes estéreo perfectamente rectificadas, de hecho, este método es ampliamente utilizado para generar mapas de disparidad.

3.2.5 Conclusión

Con los resultados obtenidos en este capítulo, imágenes perfectamente rectificadas y mapas de disparidad bastante buenos, se decide que éste es el método más apropiado para realizar la rectificación de las imágenes ya que la rectificación con OpenCV no es muy precisa y no ha dado tan buenos resultados como los triclops. Ahora ya se ha encontrado el método para rectificar las imágenes pero se presenta un problema, que es la publicación de estas imágenes rectificadas en ROS (porque en principio las librerías de ROS y triclops no son compatibles). Para resolver este problema se barajan dos posibilidades:

1. Intentar hacer que los triclops funcionen en ROS, por ejemplo haciendo links entre librerías, omitir algunas librerías que no sean necesarias y que estén causando la incompatibilidad, buscar en internet soluciones propuestas para resolver esta incompatibilidad, etc.
2. Programar una memoria compartida. Esto consiste en realizar un programa que recoja las imágenes rectificadas mediante triclops y ponerlas en memoria compartida. Una vez estén en memoria compartida y mediante sistemas de sincronismo como mutex o semáforos, se acceden a las imágenes, se rellena un mensaje con esa imagen y se publica en ROS, se libera la zona de memoria y se espera a que llegue la siguiente imagen para volver a realizar el proceso. Con esto se retrasa un poco a la hora de publicar los mensajes pero se decide sacrificar velocidad por calidad.

4 Bumblebee xb3 con triclops en ROS

Como se ha visto en el capítulo anterior, la librería oficial de la cámara estéreo Bumblebee3 es la única capaz de realizar correctamente la rectificación de las imágenes y calcular los mapas de disparidad mediante el uso de las imágenes rectificadas y el método SAD (véase el capítulo 3). En este capítulo se van a presentar los resultados de los intentos de usar los triclops directamente con ROS. Antes de ponerse directamente a programar una memoria compartida, se decide primero intentar utilizar los triclops directamente con ROS ya que sería la forma más fácil y eficaz de rectificar y publicar a la vez las imágenes en ROS. La memoria compartida tiene sus inconvenientes, por ejemplo el proceso de rectificación y publicación será un poco más lento.

Por lo tanto, primero se intenta usar los triclops directamente en ROS, programando y haciendo links de librerías y después se utiliza el package xb3, que es un driver que está disponible en internet y que intenta resolver el problema de compatibilidad Triclops-ROS.

4.1 Uso de triclops en ROS

Para utilizar los Triclops de Point Grey en ROS, primero se crea un Publisher, un pequeño programa en c++ para comprobar la compatibilidad de ROS-Triclops, pero cada vez que aparece un include, una función de triclops en un nodo ROS, da un mensaje de error de "violación de acceso" o "violación de segmento".

Se intenta resolver estos errores de varias formas, cambiando los códigos, realizando los links entre las librerías de distintas formas, ya que los triclops de Point Grey vinculan estáticamente algunas librerías que ROS también utiliza. Pero al final no se consigue utilizar directamente los triclops en ROS porque parece que hay un problema de compatibilidad, ahora el siguiente paso es intentar solucionarlo con el driver xb3 propuesto en internet. Estos problemas de compatibilidad se ilustran en **[24]** y **[25].**

4.2 Package xb3 en ROS usando triclops

El package xb3 es un driver para cámaras estéreo Bumblebee xb3 que intenta solucionar los problemas de compatibilidad Triclops / ROS. Este nodo recibe imágenes de la cámara y publica los topics y el camera_service info (servicio de información de la cámara).

Al utilizar este driver se encontraron con los mismos problemas de compatibilidad de antes, además para rectificar las imágenes necesita files (archivos) de calibración. Con este driver se puede crear un nodo en ROS, pero a la hora de ejecutarlo con la cámara en funcionamientos da el mismo tipo de error mencionado en el apartado anterior. Este driver se puede encontrar en **[26].**

4.3 Conclusión

Como se ha visto antes, para poder utilizar paquetes ROS con datos extraídos utilizando Triclops, hay que superar el problema de compatibilidad de ROS con Triclops. Este problema está causado por el hecho de que tanto la librería oficial precompilado y ROS tienen su propia versión de librería boost, lo que resulta en un error de segmentación cuando las funciones Triclops se utilizan dentro de un nodo de ROS.

Para resolver este problema de compatibilidad se decide separar el código Triclops que da acceso a las imágenes rectificadas y los mapas de disparidad del código del nodo de ROS mediante la creación de una memoria compartida.

5 Memoria Compartida

La memoria compartida es el medio más eficaz de compartir información entre programas. Un programa creará un segmento de memoria en la que otros procesos pueden acceder con la intención de proporcionar una comunicación entre ellos o evitar copias redundantes. La escritura y la lectura se hacen a una muy alta velocidad. La ventaja de la memoria compartida en comparación con un protocolo de comunicación UDP es la no limitación en el tamaño de los datos que se están transfiriendo. Un protocolo UDP tiene una limitación en el tamaño máximo de la toma de datos.

Utilizando memoria compartida, se puede transferir en tiempo real las tres imágenes rectificadas de la cámara estéreo Bumblebee xb3 entre códigos Triclops y ROS para que sean compatibles.

"*Escribir y leer de una memoria compartida es un método más rápido para intercambiar datos entre procesos, que utilizando los servicios regulares del sistema operativo. Generalmente un proceso envía datos a través de una estructura de salida que tiene que ser empaquetada (en un archivo, pipe, etc.) y luego desempaquetada en el proceso receptor. Al utilizar un área específica de memoria de forma compartida, el dato es accesible directamente a ambos procesos sin tener que solicitar gestión al sistema operativo para el transporte de los datos. Para colocar datos en la memoria compartida, el proceso emisor de datos obtiene el acceso a la zona de memoria después de haber chequeado un valor de semáforo (o "semáforo" simplemente) en estado de "permitida la escritura", escribe los datos, y entonces limpia el semáforo para indicar que hay datos disponibles para ser leídos. Al ser leídos por el proceso receptor, el semáforo es reinicializado para permitir nueva entrada de datos*." **[27]**

5.1 Programación Memoria compartida

Como se ha dicho antes, el uso de memoria Compartida es un medio muy eficaz de pasar datos entre programas. Un programa creará una porción de memoria en el que otros procesos (si están permitidos) pueden acceder.

Se crea un proceso llamado server que rectifica las imágenes con triclops y crea un segmento de memoria compartida utilizando shmget (). Se conecta este segmento de memoria compartida a un proceso llamado triclops_client, que es el nodo de ROS que recibe las imágenes rectificadas y las transforma en mensajes ROS (previamente se le da permiso a este proceso para que pueda acceder a la memoria compartida). Una vez conectado y con permiso de acceso, el nodo triclops_cliente ya puede leer la memoria compartida. El segmento de memoria compartida es descrito por una estructura de control con un identificador único que apunta a un área de memoria física. El identificador del segmento se llama el shmid. La definición de la estructura para las estructuras de control de segmento de memoria compartida se encuentran en <sys / shm.h>.

Acceso al segmento de memoria compartida

shmget () se utiliza para obtener acceso al segmento de memoria compartida. Se ha utilizado así:

int shmget(key_t key, size_t size, int shmflg);

Donde el argumento key es un valor de acceso asociado con el ID del semáforo. El argumento size es el tamaño en bytes de la memoria compartida. El argumento shmflg específica los permisos de acceso iniciales y los indicadores de control de la creación.

Conexión y desconexión del segmento de memoria compartida

Se han utilizado shmat () y shmdt () para conectar y desconectar el segmento de memoria compartida. Se han utilizan como sigue:

void *shmat(int shmid, const void *shmaddr, int shmflg);

int shmdt(const void *shmaddr);

shmat () devuelve un puntero y shmdt () desconecta el segmento de memoria compartida localizado en la dirección indicada por shmaddr.

5.2 Publicación imágenes rectificadas en Ros

Como se ha visto en capítulos anteriores, Triclops, la biblioteca oficial de la cámara estéreo Bumblebee3 es capaz de realizar la rectificación de las imágenes y calcular los mapas de disparidad mediante el uso de las imágenes rectificadas y el método de suma de diferencias absolutas (véase el capítulo 3). Como también se ha visto antes, para poder utilizar paquetes ROS con datos extraídos utilizando Triclops, se tiene que superar el problema de compatibilidad ROS / Triclops. Este problema está causado por el hecho de que tanto la librería oficial precompilada y ROS tienen su propia versión de librería boost, lo que resulta en un error de segmentación cuando las funciones Triclops se utilizan dentro de un nodo de ROS.

Para resolver este problema se ha tenido que separar el código Triclops que da acceso a las imágenes rectificadas y los mapas de disparidad del código del nodo de ROS mediante la creación de un bridge (puente) de comunicación entre ellos utilizando un protocolo de comunicación de memoria compartida. La estructura del código establecido es el siguiente:

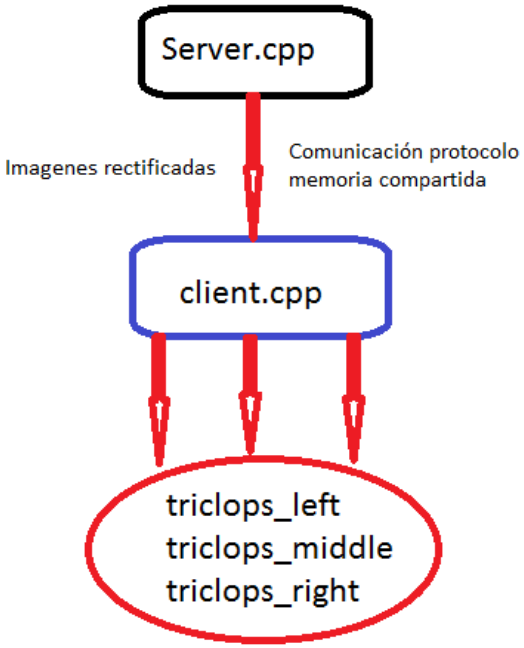


Figura 28: Estructura de comunicación entre triclops y Ros.

En la figura 28 se observa que primero se rectifican las imágenes con un código de triclops, se ponen las imágenes rectificadas en una memoria compartida, estas imágenes se pasan a un código ROS que las convierte en mensajes ROS y por último las publica.

En la siguiente imagen se puede observar el listado de nodos activos así como los topics que se están publicando:

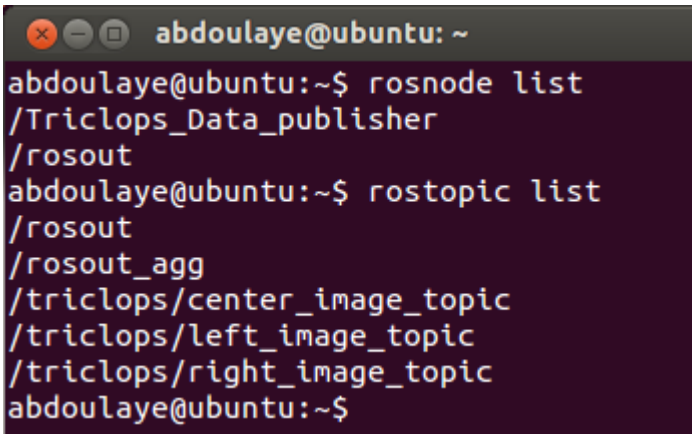


Figura 29: Rosnode list and triclops_topics.

Con esto ya se consigue unas imágenes perfectamente rectificadas con el software del fabricante y se resuelve la incompatibilidad Triclops/ROS con la memoria compartida para convertir los datos guardados en la memoria en mensajes ROS y así poder publicar estas imágenes en ROS.

6 Conclusiones y trabajos futuros

6.1 Conclusiones

El objetivo principal del proyecto era la obtención de imágenes correctamente rectificadas de la cámara Bumblebee xb3 de Point Grey, por lo que había que trabajar en las dos primeras fases de la visión estéreo, el pretratamiento y el matching para comprobar la correcta rectificación de las imágenes mediante el análisis de los mapas de disparidad wide y narrow mediante dos métodos, el método block matching de OpenCV y el método de rectificación del driver Triclops, hacer comparaciones, elegir el método más apropiado y publicar estas imágenes rectificadas en ROS.

Tras realizar las tareas anteriores se sacan las siguientes conclusiones:

- Al utilizar el método block matching de OpenCV no se obtienen mapas de disparidad adecuados, aunque la librería OpenCV cuenta con diversas funciones para el procesamiento de imágenes y constituye una gran herramienta para el desarrollo de aplicaciones en sistemas de visión artificial, algunos de sus métodos no son muy precisos.
- Utilizando el driver de Point Grey (ticlops), se consiguen unas imágenes perfectamente rectificadas y unos mapas de disparidad bastantes buenos ya que este software tiene acceso a las imágenes perfectamente rectificadas y para realizar la disparidad usa el método SAD, método de suma de diferencias absolutas.
- Para poder utilizar paquetes ROS con datos extraídos utilizando Triclops, hay que superar el problema de compatibilidad de ROS con Triclops. Este problema está causado por el hecho de que tanto la librería oficial precompilado y ROS tienen su propia versión de librería boost, lo que resulta en un error de segmentación cuando las funciones Triclops se utilizan dentro de un nodo de ROS.
- La memoria compartida es el medio más eficaz de transmitir datos entre programas, por lo que se puede resolver el problema de compatibilidad Triclops / ROS separando el código Triclops que realiza el procesamiento de las imágenes de la cámara Bumblebee xb3 del código del nodo de ROS mediante la creación un protocolo de comunicación de memoria compartida.

6.2 Trabajos futuros

- Hacer que el procesamiento de las imágenes sea continuo y seguir con la fase tres de la visión estéreo, la reconstrucción 3D.
- Utilizar las imágenes procesadas para trabajar con Point Cloud Library.
- Utilizar la cámara Bumblebee xb3 para el cálculo en el adelantamiento de otros vehículos.
- Utilización de la Bumblebee xb3 para el seguimiento del borde de la carretera.
- Utilización de la Bumblebee xb3 para la ayuda en el mantenimiento de la distancia de seguridad.

7 Presupuesto

En este capítulo se presentan los costes que han sido necesarios para la realización del proyecto. Para ello se secciona el proyecto por fases en las que se ha realizado el proyecto calculando en cada fase el tiempo invertido en la realización de las tareas. Una vez calculado el tiempo total invertido en el desarrollo del proyecto, se calculan los costes asociados a las horas de ingeniería (costes de personal).

En la Tabla 5 se presenta el tiempo total invertido en el desarrollo del proyecto, en la Tabla 6 los costes de personal, en la tabla 7 los costes de material y en la tabla 8 los costes totales.

En este proyecto a parte de la cámara bumblebee xb3 de Point Grey, solamente se ha utilizado equipamiento básico de oficina, un equipo informático con conexión a internet. Además el uso de ROS, librerías OpenCV de código abierto y gratuito y el uso de los triclops que es el software del fabricante que viene con la cámara, ha permitido aligerar los costes totales necesarios.

Fase	Descripción	Tiempo (h)
I	Estudio y compresión del problema a resolver.	10
	Iniciación a ROS realizando tutoriales.	20
II	Comprensión del código del driver xb3 con el que se va a trabajar.	10
	Procesamiento de imágenes en ROS con el driver xb3 y análisis de los mapas de disparidad.	50
	Modificación del driver xb3, procesamiento de imágenes con el driver modificado y análisis de resultados.	30
	Calibración de la cámara.	2
III	Instalación del driver privativo triclops en linux y procesamiento de imágenes con los programas originales que vienen con éste.	10
	Programación y adaptación del código de triclops a la cámara bumblebee xb3.	60
	Procesamiento de imágenes y análisis de mapas de disparidad con el driver triclops adaptado.	10
IV	Programación de memoria compartida.	20
	Programación del nodo que recibe las imágenes guardadas en la memoria compartida y las publica en ROS.	30
	SUBTOTAL 1	252
V	Revisión de códigos.	10
	Redacción de la memoria.	60
	SUBTOTAL 2	70
TOTAL		**322**

Tabla 5: Tiempo invertido en las fases del proyecto.

Suponiendo un precio horario aproximado de 25€/hora para un ingeniero novel en fase de aprendizaje, el coste de personal sería:

Concepto	Coste por unidad (€/h)	Cantidad (h)	Importe (€)
Costes asociados a las horas de ingeniería	25	322	8050
TOTAL			**8050**

Tabla 6: Costes de personal.

Concepto	Coste por unidad (€/unid.)	Cantidad (unid.)	Importe (€)
Cámara Bumblebee xb3	3000	1	3000
PC mac mini	550	1	550
TOTAL			**3550**

Tabla 7: Costes de material.

Concepto	Importe (€)
Coste de personal	8050
Coste de material	3550
TOTAL	**11600€**

Tabla 8: Costes totales del proyecto.

Nota: Estos costes se han calculado de forma aproximada, sin tener en cuenta las retenciones que se han de aplicar, por ejemplo IRPF, IVA, etc.

Anexos

A. Código Triclops_Server

En el código Triclops_Server (llamado server.cpp) se realiza el procesamiento de las imágenes y se guardan en una memoria compartida. Este código se encuentra en el siguiente enlace:

https://github.com/Dialloabdoulaye31/Shared-Memory/blob/686c7658c62b0d1ab0d52278aa6d9137f82fa040/server.cpp

B. Código Triclops_Client

En el código Triclops_Client (llamado triclops_client.cpp) se extraen las imágenes de la memoria compartida, se convierten en mensajes ROS y se publican. Este código se encuentra en el siguiente enlace:

https://github.com/Dialloabdoulaye31/Shared-Memory/blob/88503b85187905b4d4e60b03997eac0810ff45b5/triclops_client.cpp

Nota: También se puede acceder a los dos cogidos desde esta página:

https://github.com/Dialloabdoulaye31/Shared-Memory

Bibliografía

[1] 'Visión por Computador para Vehículos Inteligentes'. Juan Manuel Collado, Cristina Hilario, José María Armingol, Arturo de la Escalera. 2003.

[2] Advanced Driver Assistance Systems. http://www.ti.com/lit/sl/slyy044/slyy044.pdf, Consultado: 15 - 08 – 2014

[3] Google image, IVVI UC3M. https://www.google.es/search?q=ivvi+uc3m&biw=1366&bih=657&source=lnms&tbm=isch&sa=X&ei=ZzofVO_pJ43KaKKEgJAK&ved=0CAYQ_AUoAQ, Consultado: 15 - 08 – 2014

[4] P.J. Besl, *"Active Optical Range Imaging Sensors"*. Machine Vision and Applications, Springer-Verlag, New York, l: 127-152, 1988

[5] B.K.P. Horn, *"Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View"*. PhD thesis, Massachusetts Institute of Technology, Cambridge, 1970

[6] K. Kutulakos, S. Seitz, *"A Theory of Shape by Space Carving"*, International Journal of Computer Vision 38(3), 199–218, 2000

[7] 3D structure from 2D motion. http://www.cs.columbia.edu/~jebara/papers/sfm.pdf, Consultado: 16 - 08 – 2014

[8] D. Marr, T. Poggio, "*Cooperative computation of stereo disparity*". Science, 194:283–287, 1976

[9] D. Marr, T. Poggio, "*A computational theory of human stereo vision*". In Proceedings of the Royal Society of London B, volume 204, pages 301–328. 1979

[10] Action Recognition Robust to Background Clutter by Using Stereo Vision. http://hal.archives-ouvertes.fr/docs/00/76/86/70/PDF/ws106.pdf, Consultado: 16 - 08 – 2014

[11] M. Brown, D. Burschka, and G. Hager, "*Advances in computational stereo*". IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(8):993–1008. 2003

[12] D. Scharstein, R. Szeliski, "*Stereo matching with nonlinear diffusion*". International Journal of Computer Vision, 28(2):155–174. 1998

[13] 3D reconstruction OpenCV. http://docs.opencv.org/trunk/doc/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html , Consultado: 18 - 08 - 2014

[14] Stereo image processing in ROS. http://wiki.ros.org/stereo_image_proc , Consultado: 18 - 08 - 2014

[15] Xb3 point grey. http://ww2.ptgrey.com/stereo-vision/bumblebee-xb3, Consultado: 20 - 08 - 2014

[16] ROS introduction. http://wiki.ros.org/ROS/Introduction, Consultado: 20 - 08 - 2014

[17] OpenCV. http://opencv.org/ , Consultado: 21 - 08 - 2014

[18] cv_bridge. http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages, Consultado: 21 - 08 - 2014

[19] Stereo processing ROS. http://wiki.ros.org/stereo_image_proc, Consultado: 21 - 08 - 2014

[20] Real-Time Stereo Vision System using Semi-Global Matching Disparity Estimation: Architecture and FPGA-Implementation. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5642077 , Consultado: 22 - 08 - 2014

[21] Graph-cut-based stereo matching using image segmentation with symmetrical treatment of occlusions. http://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCEQFjAA&url=http%3A%2F%2Fwww.researchgate.net%2Fpublication%2F220307072_Graph-cut-based_stereo_matching_using_image_segmentation_with_symmetrical_treatment_of_occlusions%2Flinks%2F00b49525ca3d62d9b5000000&ei=sukfVL39GMPXas6egeAB&usg=AFQjCNHKx3mJlZj0k1DpgLk6Fqp8hfTHEQ, Consultado: 22 - 08 - 2014

[22] Point Grey Research, Triclops. http://www.ptgrey.com/products/triclopsSDK/index.asp, Consultado: 09 - 09 – 2014

[23] How to work with the bumblebee2 stereo camera on linux. http://image.diku.dk/mediawiki/index.php/BumbleBee, Consultado: 09 - 09 - 2014

[24] Triclops in ROS. http://answers.ros.org/question/124716/triclops-sdk-function-breaks-rosnodehandle/ , Consultado: 10 - 09 - 2014

[25] Triclops in ROS. http://answers.ros.org/question/66375/bumblebee-xb3-driver/, Consultado: 10 - 09 - 2014

[26] xb3.cpp. http://lars.mec.ua.pt/lartk/doc/xb3/html/index.html, Consultado: 10 - 09 - 2014

[27] 'Practica 5, sistemas informáticos en tiempo real'. José María Armingol moreno, Daniel Martín Ruano. UC3M.